



# Descrição do Design Pattern MVC e de sua aplicação no Framework Spring Boot

Ao longo das últimas décadas, o processo de desenvolvimento de software tem passado por diversas transformações. Inicialmente, tínhamos os softwares monolíticos, em que a aplicação era desenvolvida para ser executada em uma única máquina, além de ser composta por uma grande quantidade de código, ficando juntos, numa mesma camada, a lógica e regras de negócio, o tratamento de eventos e também o acesso aos dados. A seguir, num esforço para separar o código do software de acordo com a sua responsabilidade, foram propostos os modelos de camadas – comumente chamado de "n camadas", já que foram realizadas diferentes implementações dos mesmos.

Em termos teóricos, podemos dizer que o modelo de "n camadas" é um padrão de arquitetura de software (ou padrão arquitetural). O cerne desse padrão de arquitetura consiste em propor uma estruturação/organização em alto nível na construção de softwares, em que o código é agrupado de forma hierárquica em módulos, também chamados de camadas. Em linhas gerais, nessa hierarquia uma camada só poderá/deverá se relacionar com a camada imediatamente inferior. Veja na imagem abaixo um exemplo dessa divisão em camadas, no modelo de 3 camadas:



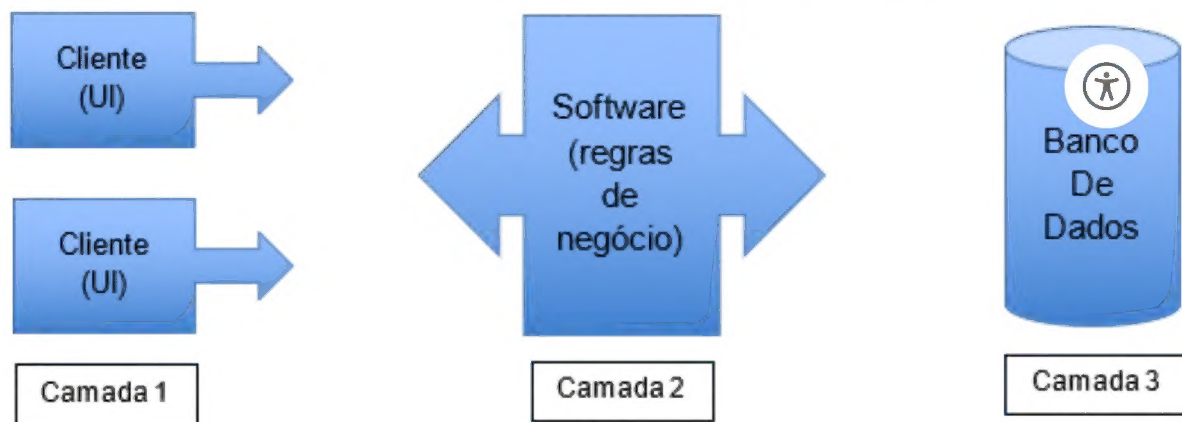



Figura 1: Modelo de 3 Camadas

Como podemos ver na imagem, no modelo em questão temos uma divisão do software em camadas:

1. A camada de acesso ou de interface, em que um ou vários clientes acessam o software;

2. Uma camada que pode ser composta internamente por vários módulos, contendo as regras e a lógica de negócio;

3. Temos ainda uma terceira camada, em que ficam armazenados os dados, ficando estes isolados do restante da aplicação.

Voltando à história da Engenharia de Software, e antes de falarmos no framework Spring Boot, cabe destacar um outro padrão arquitetural, proposto na sequência do modelo de "n camadas", por volta do final dos anos 70: o padrão MVC. Tal arquitetura foi utilizada, inicialmente, no desenvolvimento de aplicações utilizando a linguagem de programação Smalltalk-80 (uma das primeiras linguagens orientadas a objeto, além de pioneira na utilização de interfaces gráficas ricas). O MVC (sigla para Model View Controller)

propõe que as classes que compõem uma aplicação sejam divididas em três grupos:



- View: responsável pela interação com os usuários;
- Controller: tratam/respondem aos eventos gerados na camada View, contém as regras de negócio e interagem com a camada Model;
- Model: responsável pelo acesso aos dados da aplicação.

Como podemos perceber, e não por acaso, há algumas similaridades (e também confusão) quando tratamos dos modelos de três camadas e do MVC. Mais recentemente, a arquitetura MVC foi associada a um outro conceito, o de Design Patterns. Segundo Gamma (et al), podemos resumir os Design Patterns (ou Padrões de Projeto) como um meio composto por um conjunto de padrões, de auxiliar desenvolvedores a criarem software, tendo como base o conhecimento de outros arquitetos de software. Em outras palavras, os padrões de projeto são compostos por soluções para problemas comuns encontrados no processo de desenvolvimento e manutenção de software orientado a objetos. Seguindo essa linha de raciocínio, o MVC é um padrão de projeto formado, na verdade, pela aplicação/utilização em conjunto de alguns design patterns, como o Observer, Composite, Strategy, entre outros (Gamma et al).

Avançando um pouco mais na história de desenvolvimento de software, chegamos aos anos 1990/2000. Nesse ponto, com o surgimento da Web, surgiram também os sistemas Web. Embora muito semelhantes aos sistemas de "n camadas", mais precisamente aos de três camadas, tornou-se comum associar esses sistemas ao padrão MVC, uma vez que alguns frameworks de desenvolvimento, nascidos na mesma época em questão, adotaram os conceitos e

termos típicos desse padrão em suas implementações. Um dos frameworks mais utilizados atualmente, e que faz uso dos conceitos do Design Pattern MVC, é o Spring.

O Spring é um framework open source, cujos fundamentos foram definidos por seu criador, Rod Johnson, em 2004, em um livro que tratava de design e desenvolvimento utilizando Java (J2EE). Tal framework se baseia em padrões como Inversão de Controle e Injeção de Dependências. Além disso, é um framework que se diferencia de outras soluções, como o próprio J2EE, por permitir, dada a sua composição modular, a implementação de soluções de forma mais enxuta, sem a necessidade de códigos e componentes adicionais que, por muitas vezes, acabam nem sendo utilizados. Nesse contexto, de ser um framework modular, são disponibilizados vários componentes como: Spring Data, Spring Cloud, Spring Security, entre outros. A partir desse ponto trataremos, de forma mais aprofundada, do Spring Boot – em particular sobre as suas camadas de separação de código, a exemplo do que vimos anteriormente no Padrão MVC.

O ponto de partida indicado para aprender mais sobre o Spring Boot e começar a desenvolver utilizando tal framework é o seu site oficial: https://spring.io/projects/spring-boot. **(Acesso em 27/09/2022)** Nele é possível entender melhor o seu funcionamento, conhecer os componentes disponíveis e também codificar seguindo alguns tutoriais. Como já mencionado, o Spring Boot faz uso da separação em camada dos códigos, organizando-os de acordo com suas responsabilidades. Tal organização é bastante similar à que vimos ao tratarmos do MVC, com a diferença de que, no Spring Boot, há mais camadas disponíveis: Entity, Repository, Service e Controller.

Antes de falarmos, especificamente, sobre cada uma das camadas acima, é importante frisar que o tipo mais comum de aplicações desenvolvidas com o Spring Boot (embora não as únicas possíveis) são as APIs Restful. Esse tipo de aplicação é caracterizado por uma interface através da qual dois sistemas podem trocar informações entre si, ou até mesmo como outros tipos de cliente podem trocar informações com um determinado sistema. Essa conectividade entre sistemas é importante, uma vez que um sistema pode fazer uso de recursos disponibilizados por outro sistema, sejam esses recursos compostos por funcionalidades – como cálculo de fretes, por exemplo, ou por dados necessários ao negócio, como informações de endereço a partir de um número de CEP. Além das já mencionadas, é importante conhecermos, nesse ponto, algumas outras características importantes de uma API Restful:

- Os dados são trafegados no formato JSON;
- A API utiliza o protocolo HTTP para a transmissão dos dados – incluindo os verbos HTTP, de acordo com a operação a ser realizada (GET, POST, PUT, etc.);
- Uma API não possui interface gráfica.

A última característica acima nos remete ao que vimos anteriormente sobre o padrão MVC, uma vez que a camada V (View) é composta, justamente, por uma interface. Logo, no padrão de camadas utilizado pelo Spring Boot, não há uma camada responsável pelas classes de integração com o usuário. Com isso, a camada Controller exerce o papel de ser o ponto de entrada da API. No Controller ficam todos os recursos (ou endpoints) disponibilizados pela API. Essa camada não deverá conter regras de negócio ou códigos de acesso a dados. A responsabilidade de conter as regras de negócio fica a cargo da

camada Service. Essa, por sua vez, não realiza acesso a dados ou disponibiliza os recursos da API. Por fim, temos ainda as camadas Entity e Repository. Essas camadas, juntas, ao pensarmos no MVC, equivaleriam à camada Model. Nelas temos, na Entity, nossas classes que representam (através da utilização de ORM – Mapeamento Objeto Relacional) as entidades existentes no banco de dados utilizado na API. Já no Repository ficam as instruções SQL, as queries, tanto nativas – herdadas do JPA (Java Persistence API), como outras customizadas pelo desenvolvedor. Através da imagem abaixo você poderá visualizar essa organização em camadas do Spring Boot:

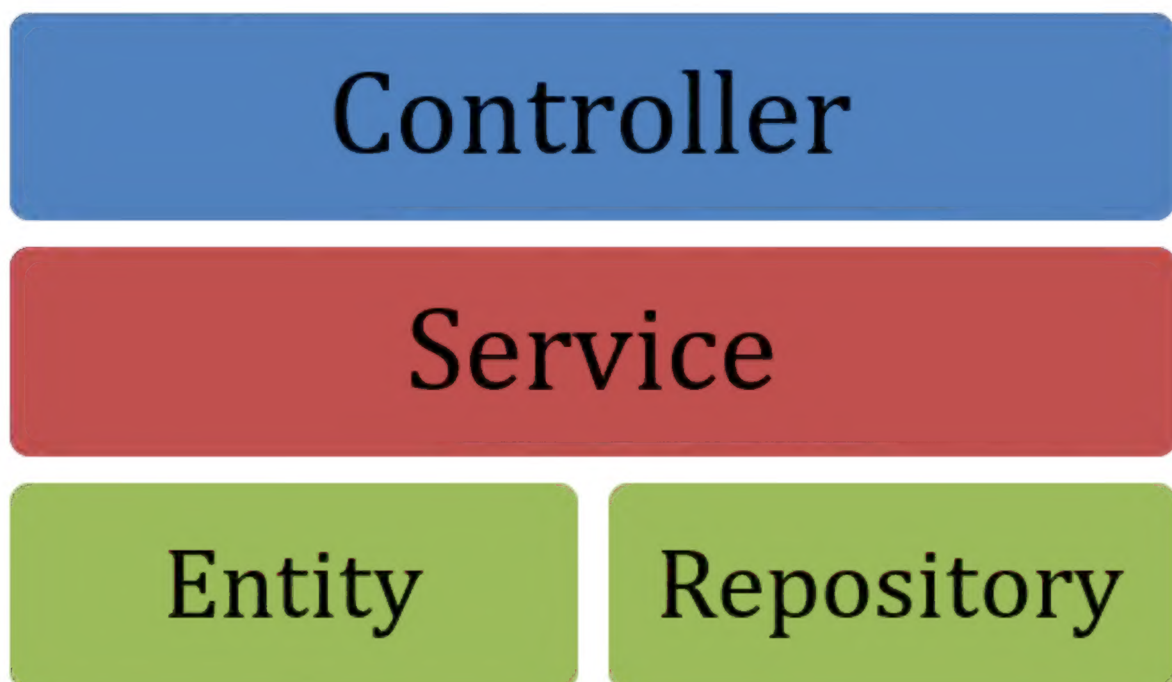



Figura 2: Organização em Camadas do Spring Boot

Cabe destacar que essa organização em camadas do Spring, em termos de implementação, deve ser hierárquica. Embora o Spring não impeça que classes e métodos que fazem parte do Controller

acessem classes e métodos disponibilizados na camada Entity, é importante, no momento da implementação, manter o acesso restrito e hierárquico entre as camadas. Com isso, a camada Controller sempre se relacionará com a camada Service. Essa, por sua vez, se relacionará com a camada Repository. Por último, as classes que ficam na camada Entity são usadas para o tráfego de dados em toda a API. Vejamos, nas três imagens a seguir, um exemplo prático do fluxo de requisição e resposta em uma API implementada com o Spring Boot:

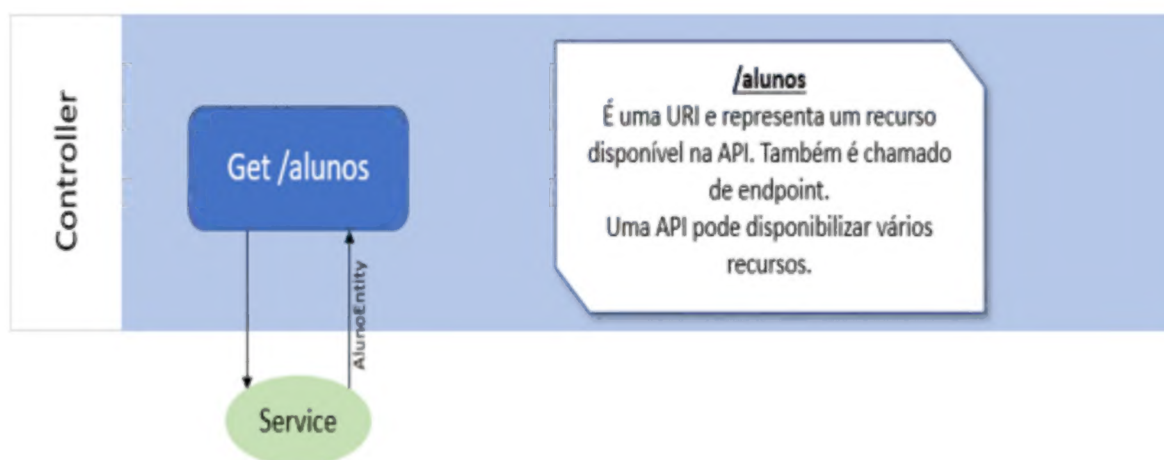



Figura 3: Fluxo de Requisição e Resposta numa API escrita com Spring Boot – Controller

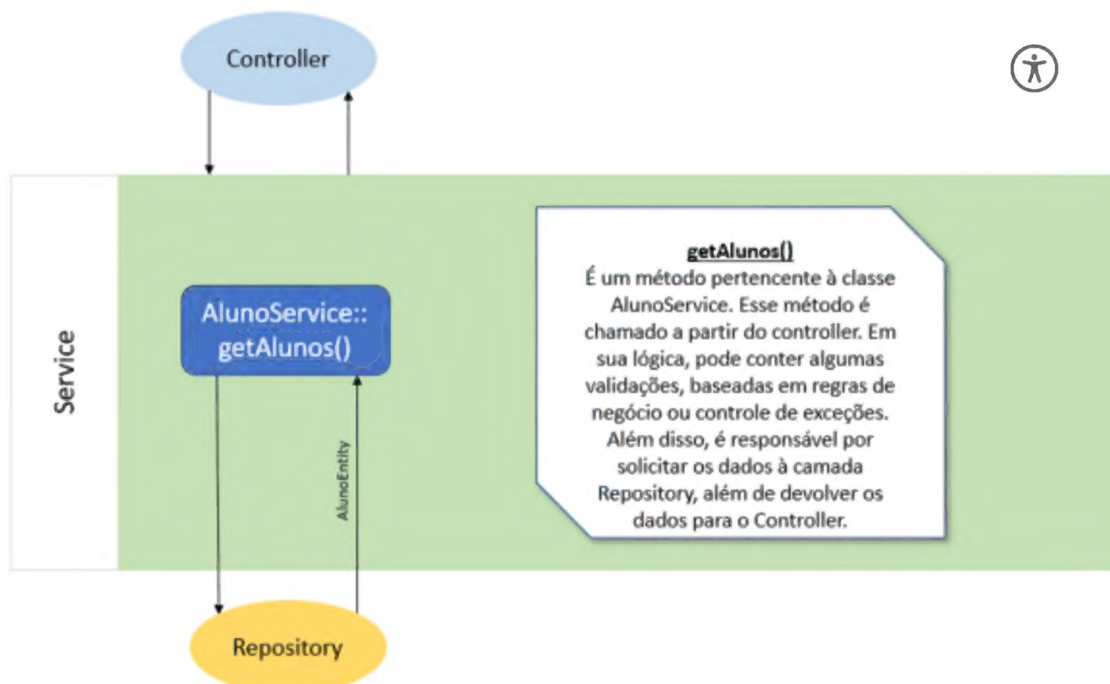



Figura 4: Fluxo de Requisição e Resposta numa API escrita com Spring Boot – Service

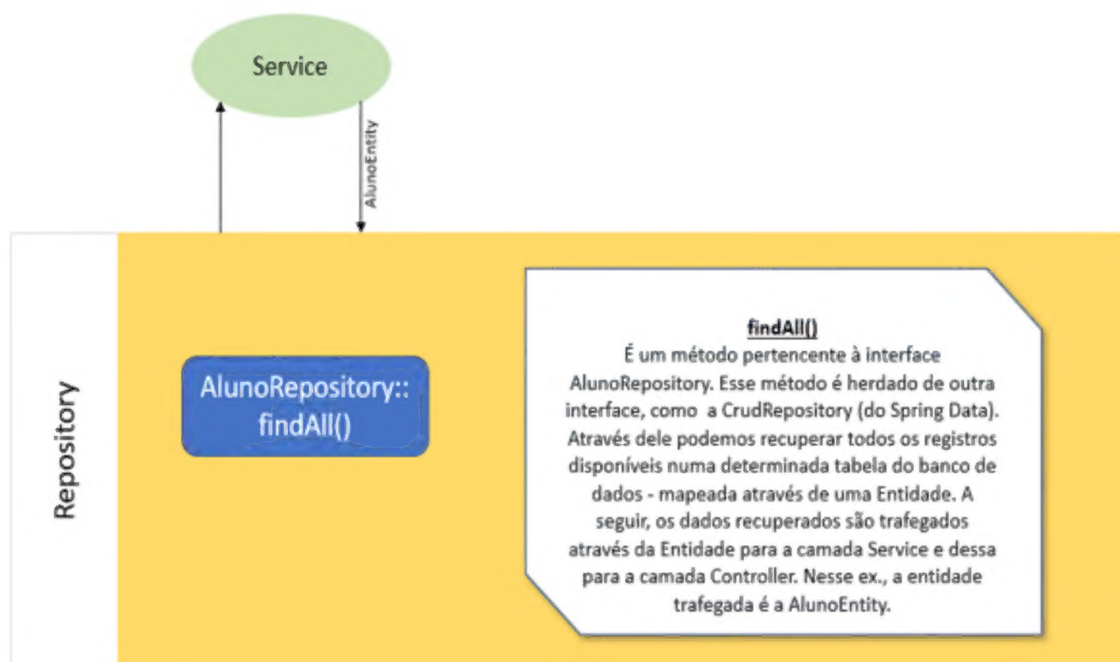



Figura 5: Fluxo de Requisição e Resposta numa API escrita com Spring Boot – Repository



Ao longo desse módulo foram descritos alguns padrões de projeto, como o Modelo de 3 Camadas e o Design Pattern MVC. Além disso, aprofundando os conceitos de separação de camadas/responsabilidades, foi demonstrada a organização em camadas do framework Spring Boot para a construção de uma API Restful.

**Atividade Extra**

Se quiser entender em detalhes o funcionamento do Spring Boot, assim como visualizar outras ferramentas disponíveis no framework, acesse o link abaixo:

Link: https://spring.io/projects/spring-boot **(Acesso em 14/09/2022)**

**Referência Bibliográfica**

GAMMA, Erich et al. **Padrões de projetos: soluções reutilizáveis de software orientado a objeto.** Porto Alegre: Bookman, 2007.

JOHNSON, Rod. **Expert One-on-One J2EE Design and Development**. Indianapolis: Wiley Publishing, 2003.



## Atividade Prática Módulo 1

**Título da Prática:** Criar a estrutura em camadas de uma API Restful

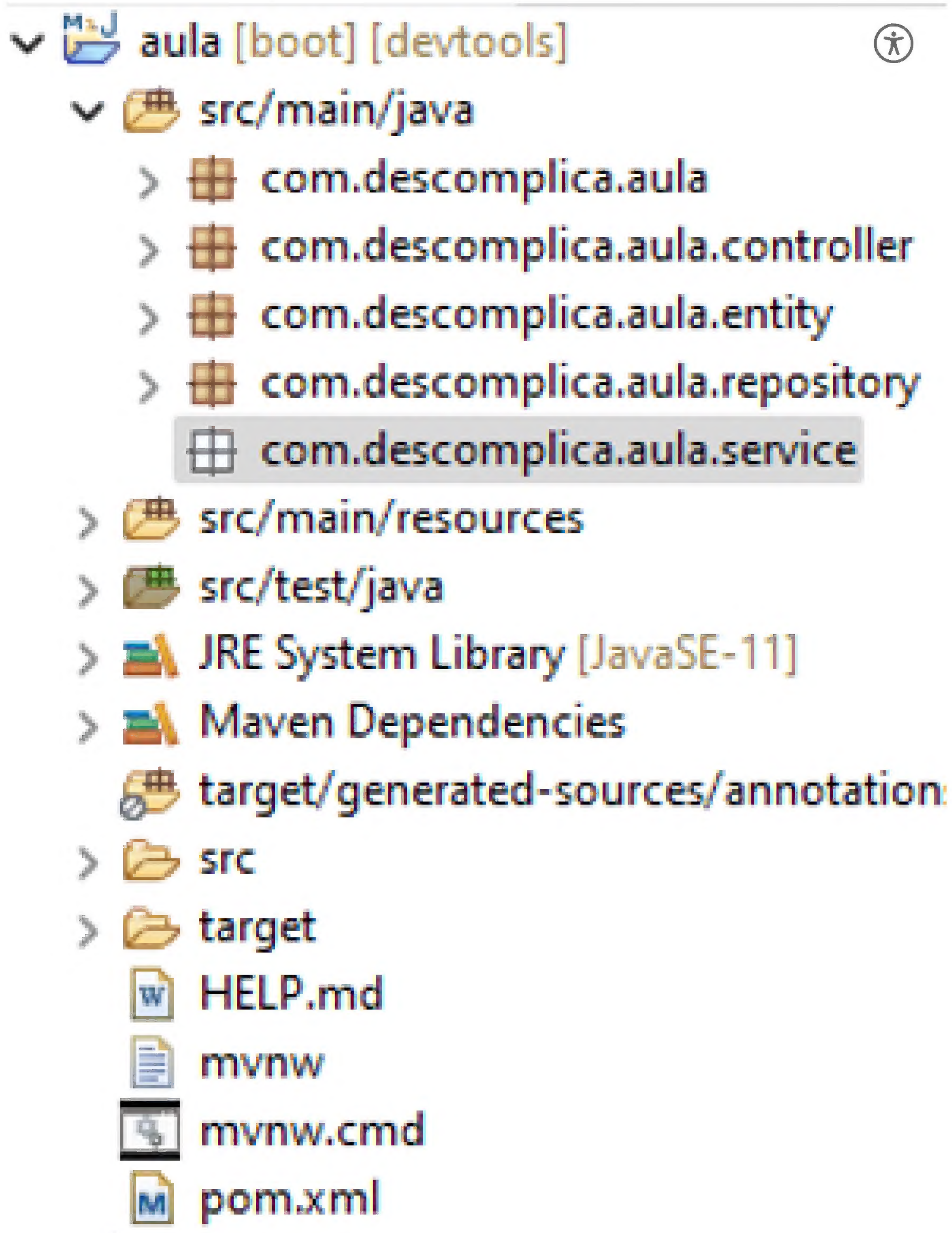
**Objetivos:** Analisar o entendimento do aluno sobre a separação em camadas, conforme responsabilidades, de uma API Restful.

**Materiais, Métodos e Ferramentas:** IDE (Spring Tool Suite; Jetbrains Intellij; etc)

## Atividade Prática

Uma API Restful desenvolvida com o framework Spring Boot deve seguir a organização proposta pelo mesmo, onde os códigos são separados em diferentes camadas, de acordo com a sua responsabilidade. Nesse sentido, crie o esqueleto de uma API Restful, organizando os códigos nos packages: controller, service, entity e repository:

## Gabarito - Atividade Prática




Na imagem acima é possível visualizar o esqueleto (a estrutura padrão) de uma API Restful codificada, segundo a organização proposta pelo Spring Boot. Tal imagem se refere ao print do projeto configurado na IDE Spring Tool Suite (STS). Independente da IDE utilizada, a estrutura deverá ser semelhante a esta.

**Ir para exercício**

